Game loop and time

Piotr Fulmański

Wydział Matematyki i Informatyki,
Uniwersytet Łódzki, Polska

8 października 2015

Table of contents

Game loop

A traditional game loop is broken up into three distinct phases

- processing inputs,
- updating the game world,
- and generating outputs.

Game loop

At a high level, a basic game loop might look like this:

```
while game is running
  process inputs
  update game world
  generate outputs
loop
```

Take into account very common part of the game code

```
// Update x position by 5 pixels
enemy.position.x += 5
```

X MHz $\rightarrow$ 150px
XX MHz $\rightarrow$??? 300px ???
Wow! It's to fast to play!
To solve this issue, we need to introduce the concept of **delta time**: *the amount of elapsed game time since the last frame*.

Take into account very common part of the game code

```
// Update x position by 5 pixels
enemy.position.x += 5
```

X MHz $\rightarrow$ 150px
XX MHz $\rightarrow$??? 300px ???
Wow! It's to fast to play!
To solve this issue, we need to introduce the concept of **delta time**: *the amount of elapsed game time since the last frame*.

Take into account very common part of the game code

```
// Update x position by 5 pixels
enemy.position.x += 5
```

X MHz $\rightarrow$ 150px
XX MHz $\rightarrow$??? 300px ???
Wow! It's to fast to play!
To solve this issue, we need to introduce the concept of **delta time**: *the amount of elapsed game time since the last frame*.

Take into account very common part of the game code

```
// Update x position by 5 pixels
enemy.position.x += 5
```

X MHz $\rightarrow$ 150px
XX MHz $\rightarrow$??? 300px ???
Wow! It's to fast to play!
To solve this issue, we need to introduce the concept of **delta time**: *the amount of elapsed game time since the last frame.*

Take into account very common part of the game code

```
// Update x position by 5 pixels
enemy.position.x += 5
```

X MHz $\rightarrow$ 150px
XX MHz $\rightarrow$??? 300px ???
Wow! It's to fast to play!
To solve this issue, we need to introduce the concept of **delta time**: *the amount of elapsed game time since the last frame*.

Use delta time

In order to convert the preceding pseudocode to use delta time, we need to think of the movement **not in terms of pixels per frame, but in terms of pixels per second**.

Use delta time

So if the ideal movement speed is 150 pixels per second, this pseudocode would be preferable

```
// Update x position by 150 pixels/second
enemy.position.x += 150 * deltaTime
```

Use delta time

Now the code will work perfectly fine regardless of the frame rate.

- At 30 FPS, the enemy will move 5 pixels per frame, for a total of 150 pixels per second.
- At 60 FPS, the enemy will only move 2.5 pixels per frame, but that will still result in a total of 150 pixels per second.

The movement certainly will be smoother in the 60 FPS case, but the overall per-second speed will be identical.

Use delta time

Now the code will work perfectly fine regardless of the frame rate.

- At 30 FPS, the enemy will move 5 pixels per frame, for a total of 150 pixels per second.
- At 60 FPS, the enemy will only move 2.5 pixels per frame, but that will still result in a total of 150 pixels per second.

The movement certainly will be smoother in the 60 FPS case, but the overall per-second speed will be identical.

Use delta time

But how do you calculate what the delta time should be every frame?

- First, the amount of real time that has elapsed since the previous frame must be queried.
- Once the elapsed real time is determined, it can then be converted to game time. Depending on the state of game, this may be identical in duration or it may have some factor applied to it.

Use delta time

But how do you calculate what the delta time should be every frame?

- First, the amount of real time that has elapsed since the previous frame must be queried.
- Once the elapsed real time is determined, it can then be converted to game time. Depending on the state of game, this may be identical in duration or it may have some factor applied to it.

Use delta time

```
realDeltaTime = 0.0f;
lastUpdateTime = GetCurrentTime();
gameTimeFactor = 1.0;

while game is running
  realDeltaTime = GetCurrentTime() - lastUpdateTime
  lastUpdateTime += realDeltaTime
  gameDeltaTime = realDeltaTime * gameTimeFactor

  GrabInput();
  UpdateGame(gameDeltaTime);
  RenderGame();

loop
```

Problems with game delta time

Although it may seem like a great idea to allow the simulation to run at whatever frame rate the system allows, in practice there can be several issues with this. Most notably, any game that has even basic physics (such as a platformer with jumping) will have wildly different behavior based on the frame rate. This is because of the way numeric integration works, and can lead to oddities such as characters jumping higher at lower frame rates. Furthermore, any game that supports online multiplayer likely will also not function properly with variable simulation frame rates.

Issues with variable time steps

Once you are using numeric integration, you more or less cannot use variable time steps. That's because the accuracy of numeric integration is wholly dependent on the size of the time step. The smaller the time step, the more accurate the approximation.
This means that **if the time step changes from frame to frame, the accuracy of the approximation would also change from frame to frame**. If the accuracy changes, the behavior will also change in very noticeable ways.

Solution (incorrect)

The simplest solution is to implement frame limiting, which forces the game loop to wait until a target delta time has elapsed.
For example, if the target frame rate is 30 FPS (a new frame is generated every 33ms) and only 20ms has elapsed when all the logic for a frame has completed, the loop will wait an additional 13.3ms before starting its next iteration.

Solution (incorrect)

```
realDeltaTime = 0.0
lastUpdateTime = GetCurrentTime()
gameTimeFactor = 1.0
targetFrameTime = 33.3 // 33.3ms for 30 FPS

while game is running
  realDeltaTime = GetCurrentTime() - lastUpdateTime
  lastUpdateTime += realDeltaTime
  gameDeltaTime = realDeltaTime * gameTimeFactor

  GrabInput()
  UpdateGame(gameDeltaTime)
  RenderGame()

  while (time spent this frame) < targetFrameTime
    // Do something to take up a small amount of time
    ...
  loop
loop
```

Problems with this solution

This solution does not guarantee that game world would be updated with constant rate. Imagine that update for some frame took

- less than `targetFrameTime`. In this case inner `while` would be executed few times to consume free time. Thanks to this, next iteration of main `while` should start (more or less) exactly when `targetFrameTime` elapse. So we would update game world with time step equal to `targetFrameTime`
- more than `targetFrameTime`. Then `realDetlaTime` would be greater than `targetFrameTime` and we would update game world with time step different (greater) than `targetFrameTime`

Problems with this solution

```
realDeltaTime = 0.0
lastUpdateTime = GetCurrentTime()
gameTimeFactor = 1.0
targetFrameTime = 33.3 // 33.3ms for 30 FPS

while game is running
  realDeltaTime = GetCurrentTime() - lastUpdateTime
  lastUpdateTime += realDeltaTime
  gameDeltaTime = realDeltaTime * gameTimeFactor

  GrabInput()
  UpdateGame(gameDeltaTime) !!! NOT CONSTANT !!!
  RenderGame()

  while (time spent this frame) < targetFrameTime
    // Do something to take up a small amount of time
    ...
  loop
loop
```

Problems with this solution

The main problem is that given solution focus on correct „frame rate" for graphics but not physics. We can say that „frame rate" for physics is the same as for graphics. Yes, it's true but the length of delta time used by `UpdateGame` method – physic method – varies form frame to frame.

Solution (correct)

```
realDeltaTime = 0.0
lastUpdateTime = GetCurrentTime()
gameTimeFactor = 1.0
targetFrameTime = 33.3 // 33.3ms for 30 FPS; this is our TIME ST
accumulator = 0.0

while game is running
  realDeltaTime = GetCurrentTime() - lastUpdateTime
  lastUpdateTime += realDeltaTime
  accumulator += realDeltaTime
  gameDeltaTime = realDeltaTime * gameTimeFactor

  GrabInput()
  while (accumulator > targetFrameTime)
    UpdateGame(targetFrameTime) !!! NOW THIS IS CONSTANT !!!
    accumulator -= targetFrameTime
  loop
  RenderGame()
loop
```